COMPARATIVE PERFORMANCE EVALUATION OF THREE VOTING SCHEMES*

W. Najjar and J-L. Gaudiot†

Information Sciences Institute
University of Southern California
Marina del Rey, California

†Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California

Abstract

0* This material is based upon work supported in part by DARPA under NASA cooperative agreement NCC 2-539

1 Introduction

Fault-tolerant systems aim at providing correct computational results in the presence of faults as well as system-wide diagnosis. This is achieved through the use of various redundancy techniques. In general, redundancy techniques call for the multiple execution of a computation, the majority results is then taken to be the correct result. Redundancy techniques can be applied either in *time* or *space*. In time redundancy, the same computation is executed repetitively until a majority result is obtained [PF82]. While this technique eliminates the effects of intermittent or transient faults, permanent faults cannot be detected unless different hardware units are used for the different executions. Space redundancy is commonly known as the NMR scheme, where N copies of the computation are executed on N hardware elements, N being an odd number. Variants of the NMR scheme, known as static or dynamic redundancy, are described in [BF76, SS82].

Multicomputer and multiprocessor systems have an inherent redundancy that can be exploited to provide fault-tolerant operations. Preparata *et al.* [PMC67] proposed a scheme where diagnosis methods are used to detect faulty elements that are subsequently isolated from the system. Based on this scheme, Kuhl and Reddy introduced the notion of *distributed fault-tolerance* which is the ability of a system to detect and isolate faulty elements without relying on any central controller or memory system [KR80]. For a survey of fault-tolerance issues in large-scale systems the reader is referred to [KR86].

The hardware redundancy available in multiprocessor and multicomputer systems can be exploited not only to acheive a faster execution but also a system reliablity much larger than that of a single element as described in [vN56]. Assuming that any computation task can always be retried on any processing element, then redundancy techniques can be applied in both time and space. Such a scheme to achieve fault-tolerant operations in distributed systems was proposed by Agrawal in [Agr85] this scheme will be described in more details in the next section.

Any scheme that exploits redundancy to achieve a more reliable execution has to rely on some form of voting. The objective of this paper is to compare and evaluate the performance of three such voting schemes: (1) a variant of the classical triple-modular redundancy, (2) the recursive scheme rpoposed by Agrawal [Agr85] and (3) an iterative dual-modular redundancy technique.

The rest of the paper is organized as folows: the three algorithms a

re described in details in section 2. Section 3 provides their analysis and performance evaluation. The results and their implications are discussed in section 4. The conclusions are presented in section 5.

2 Three Voting Schemes

In this section we describe the three voting schemes under consideration. All three schemes are based on a small set of assumptions about both the computation and the system models. These assumptions are:

- *Task based computation:* a computation is modeled as a set of independent, and possibly concurrent, tasks.
- *Task retriability:* all tasks are assumed to be fully retriable.
- *Failure modes:* each processors has a probability p of being not faulty. There are n possible and equally probable failure modes a processor can be in. In other words there are n possible wrong results a faulty processor can produce with a probability of $\frac{1-p}{n}$ each.
- *Failure-free voter:* in the subsequent analysis we will not consider the effects of voter failures.

2.1 Triple-Modular Redundancy

TMR is without doubts the oldest known voting scheme and the most studied and utilized in highly reliable systems. In this paper we will be considering a variant of TMR. Each task is executed on three distinct processors concurrently. The results are compared. The majority result becomes the final outcome. If no consensus is reached on a majority, i.e. all three results are different, then the process is once again repeated on another set of three processors. the outcome at every retry is therefore independent of the previous results. In the TMR method the final outcome is a *majority vote* of two out of three.

2.2 A Recursive Algorithm

The recursive algorithm for fault-tolerance (RAFT) was initially described by Agrawal in [Agr85]. Its implementation and experimental results were reported in [AA86]. The algorithm consists in initially executing every task concurrently on two processors. The two results are then compared. In case of a mismatch the same task is executed again on a third processor and that result compared to the previous two. The re-execution of a task on a new processor is repeated until the last result matches any one of the previous results. Upon a match the final outcome would therefore be a *plurality vote* of two out of L where L is the total number of processors employed. This method requires a *result history* to be preserved and carried from every task retry to the next one until a match is achieved.

2.3 An Iterative Algorithm

This scheme can be seen as a hybrid of the previous two. Each task is executed concurrently on two processors and the two results are compared. In case of a mismatch the task is retried again on another set of two processors. At every retry only the last two results are compared. The final outcome is therefore a *unanimity vote* of two out of two. In the rest of this paper this algorithm will be referred to as DMR for dual modular redundancy. Unlike the recursive algorithm DMR does not require a result history to be preserved. Like TMR it relies on independent trials.

3 Performance Evaluation

In all three schemes, at each retry of a task there are three possible outcomes as shown in Figure 1:

- a *match*, which can be either:
  - *correct* with probability P_c, or
  - *incorrect* with probability P_i.
- *no-match.*

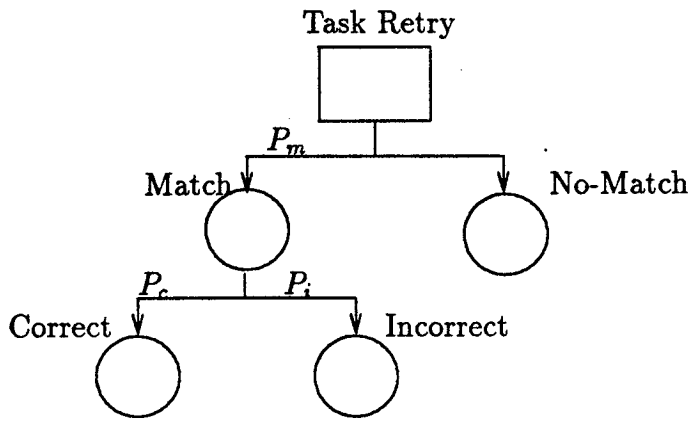

Figure 1: Possible outcomes of a task retry

As stated in the previous section, each processor is assumed to be non-faulty with a probability p. A faulty processor produces a result that can have n possible values. One can think of the result as being a k bits word and therefore there is one correct result and $n = 2^k - 1$ possible erroneous ones.

3.1 Performance Measures

The following performance measures will be used in comparing the three schemes. All of these measure will be evaluated as functions of p and n.

- *Number of iteration:* $I(p)$, is the expected number of times a task will be retried on one or more processors. It is therefore also a measure of the time necessary to achieve a match.
- *Number of processors used:* $C(p)$, is a measure of the amount of resources used.
- *Probability of a match:* $P_m = P_c + P_i$.
- *Quality of the decision:* $QD(p)$, is the conditional probability of a match being correct given that a match occurred.

$$QD = \frac{P_c}{P_m}$$

- *Quality of the decision improvement factor:* $QDIF(p)$, is a measure of the improvement a scheme provides over a single processor execution.

$$QDIF(p) = \frac{1-p}{1-QD(p)}$$

- *Relative improvement:* measures the improvement on $QDIF(p)$ per unit of cost overhead. We will consider two types of overhead: time and resource utilization. The relative improvement per time, Rit, is measured as the fraction of $QDIF$ per task retry

$$Rit(p) = \frac{QDIF(p)}{I(p)}$$

The relative improvement per resources utilized is the fraction of $QDIF$ per processor used to reach a match.

$$Rir(p) = \frac{QDIF(p)}{C(p)}$$

All of the above measures can be derived from the probabilities P_c and P_i. These probabilities are derived analytically in the next section for all three schemes.

3.2 Analytical Derivations

All three schemes are evaluated using these measures in the rest of this section. To distinguish them, these measures will be superscripted by M, P and U for *Majority*, *Plurality* and *Unanimity* respectively.

The Majority Vote Scheme A correct match is obtained when either all three or two of the processors agree, therefore

$$P_c = p^3 + 3p^2(1-p)$$

An incorrect match can occur when either two processors agree on an incorrect value with a probability of $1/n$ or all three of them with probability $1/n^2$. Therefore

$$P_i = 3p(1-p)^2\frac{1}{n} + (1-p)^3\frac{1}{n^2}$$

The expected number of iterations is

$$I = \sum_{i=1}^{\infty} iP_m(1-P_m)^{(i-1)} = \frac{1}{P_m}$$

Three processors are employed at every iteration, therefore the number of processors used is $C = 3I$.

The Plurality Vote Scheme [1]

In this scheme a match is obtained at the i^{th} iteration (which produces the $(i+1)$ result) if the last result matches any of the preceding ones. Unlike the other two schemes, the probability measures here are function of the number of results already obtained (denoted by l).

Let $Q_1(l)$ be the probability of having all l results incorrect and distinct (i.e. no match) and $Q_2(l)$ that of having only one correct result and $(l-1)$ incorrect ones.

$$Q_1(l) = \frac{n!}{(n-l)!}\frac{(1-p)^l}{n^l}$$

$Q_2(l)$ can be derived from Q_1 as

$$Q_2(l) = lpQ_1(l-1)$$

From these two measures we can derive P_c and P_i as

$$P_c(l) = pQ_2(l-1)$$

$$P_i(l) = \frac{(1-p)(l-1)}{n}Q_1(l-1) + \frac{(1-p)(l-2)}{n}Q_2(l-1)$$

The expected number of processors used is equal to the expected number of results

$$C = \sum_{l=2}^{\infty} l(P_i(l) + P_c(l))$$

Every iteration subsequent to the first one employs only one additional processor, therefore the expected number of iterations is $I = C - 1$.

[1] The analysis of this scheme is reproduced and summarized from Agrawal in [Agr85]

Since both P_c and P_i and therefore P_m are functions of the number of processors l then so is QD. We can express the quality of the decision at or before the L^{th} result as

$$QD(L,p) = \frac{\sum_{l=2}^{L} P_c(l,p)}{\sum_{l=2}^{L} P_m(l,p)}$$

In evaluating QD we will use the expected value of L which is C.

The Unanimity Vote Scheme The evaluation of the probabilities for this scheme is very similar to that of the TMR.

$$P_c = p^2$$

$$P_i = (1-p)^2 \frac{1}{n}$$

As in the TMR case $I = \frac{1}{P_m}$ and $C = 2I$. Each iteration employs two processors, therefore $C = 2I$.

3.3 Comparisons of the Schemes

The comparison is based on the measure of the expected number of iterations $I(p)$, the quality of the decision improvement factor $QDIF(p)$ and the relative improvements $Rit(p)$ and $Rir(p)$. For the sake of realism, in comparing the three schemes we will consider only values of p in the range $[0.45, 1.0[$.

The plots of $I(p)$ are shown in Figure 2. These plots show that the TMR scheme requires much less reties than the other two schemes. For lower values of p ($p < 0.85$) the recursive algorithm has a better performance than the iterative one. However, as p increases the values of I^P and I^U converge. As expected, all three values converge to $I = 1$ as $p \Longrightarrow 1.0$.

Figure 3 shows the plots for $QDIF(p)$. For both the RAFT and DMR algorithms, the $QDIF(p)$ is substantially larger than that of the TMR. It is interesting to note the inflection point on the $QDIF^P$ between the values of $p = 0.65$ and 0.7. For $p > 0.7$ the recursive and the iterative algorithms have the same values of $QDIF$.

The plots for the relative improvements in per time and resource overhead are shown in Figures 4 and 5 respectively. Although the TMR scheme was shown to require substantially smaller number of iterations to reach an

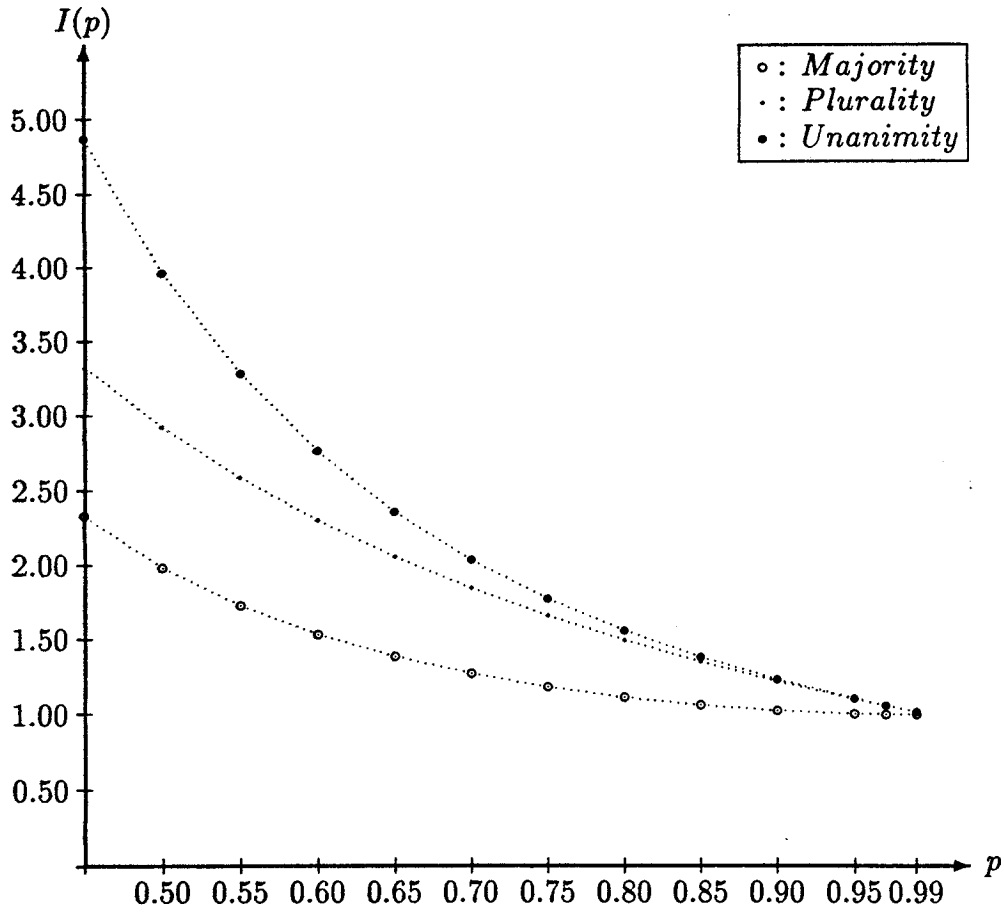

Figure 2: Expected number of iterations

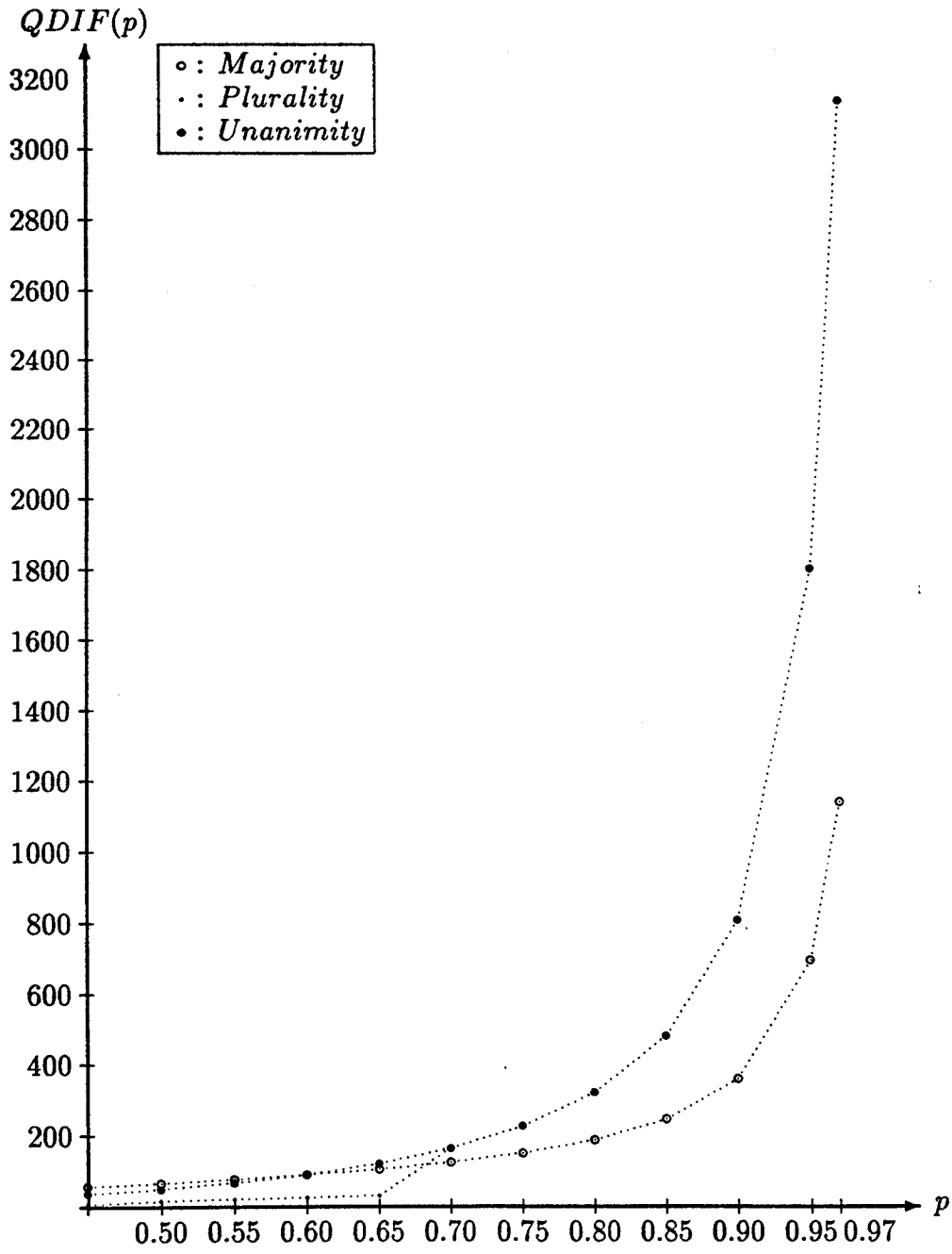

Figure 3: Quality of the decision improvement factor

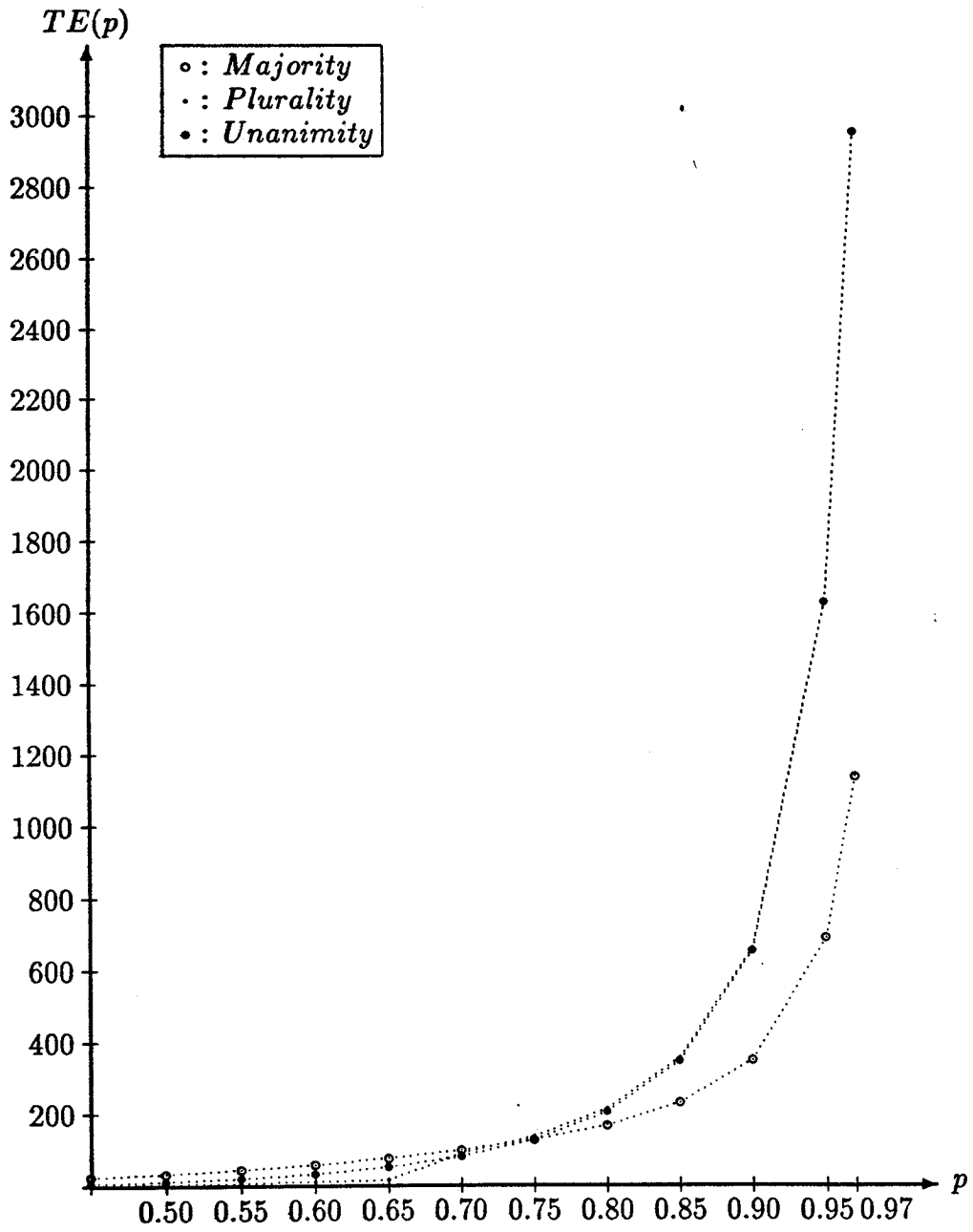

Figure 4: Time Efficiency

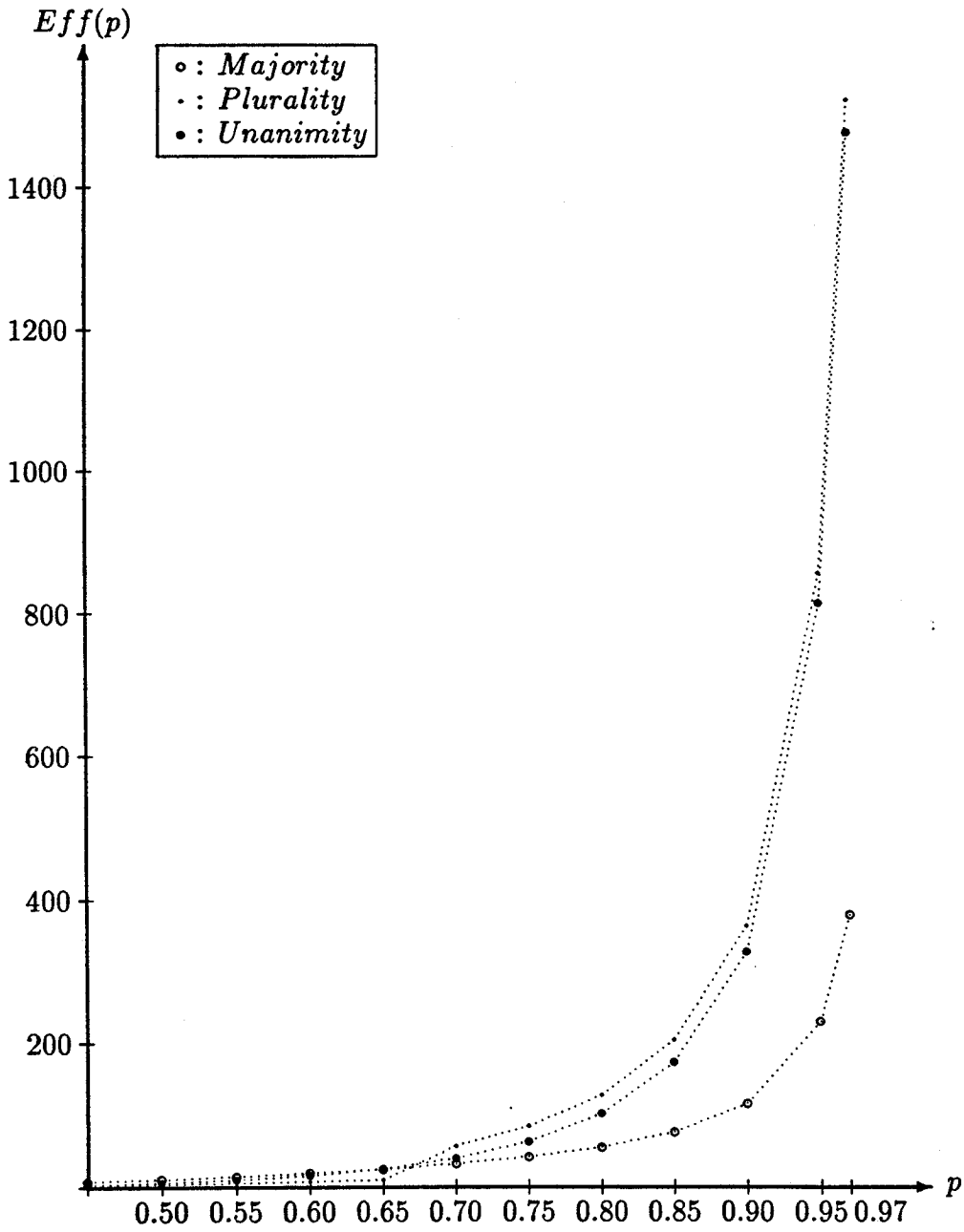

Figure 5: Resource Efficiency

agreement, the plots in Figure 4 show that the improvement achieved per time to execute a task is still lower than the other two schemes. As in the plots for $QDIF(p)$, both RAFT and DMR have the same values of $Rit(p)$ for larg values of p. In Figure 5, for lower values of p ($p < 0.65$) the TMR and DMR have very close values of $Rir(p)$ and are both larger than that of the recursive algorithm. For larger values of p RAFT and DMR have increasingly better efficiencies than the TMR and their respective values are very close differing by a small and constant amount.

4 Discussion of Results

The results of the preceding section show that for larger values of p ($p > 0.85$) the probability of getting a *correct match* using the either the recursive or the iterative algorithms is substantially higher than with the TMR. The TMR, however, requires a smaller number of iterations to reach an agreement.

Intuitively this phenomenon can be explained as follows: at the first iteration, for both RAFT and DMR the choice for a match are *two out of two* while for TMR it is *two out of three.* Therefore the TMR offers more possibilities for an incorrect match. After the second retry in RAFT (i.e. when there are three results) the situation is identical to that of TMR with only one iteration. They have therefore the same probabilities of a correct or incorrect match. When the three results are mismatched the next retry would provide RAFT with a choice of *two out of four* while the TMR will once again be a choice of two out of three. If we assume reasonable values of p (i.e. $p > 0.5$) and a large value for n its is more probable to get a correct match of two out of four than of two out of three.

The DMR algorithm is identical to the TMR except that it uses two processors instead of three at every task retry. This limitation on the number of processors increases the expected number of iterations to reach an agreement. However its vote is a two out of two choice. Given the assumptions on p and n, the likelihood of an incorrect match is therefore much lower than that of the TMR.

The RAFT and DMR algorithms were shown to provide, for all practical purposes, the same performance levels. RAFT, on the other hand is the least wasteful of system resources of all three schemes. It requires a smaller number of retries than DMR and each retry would use only on e additional processors.

The RAFT algorithm, however, requires that a history of the task execution be maintained and communicated to every processor performing a retry on that task. The DMR algorithm, like the TMR, is based on independent retries and would not require such an overhead.

In the preceding section the performance of the three algorithm was evaluated for values of p up to 0.97. These plots are useful to show the overall trend in performance over a wide range of values for p. In most cases however, the actual or expected reliability of a single processor is much larger than 0.97. For illustration purposes, Table 1 shows the same performance measures for higher processor reliability ($p = 0.999$). The values show the RAFT and DMR algorithms to have the same performance levels. A retry in the TMR scheme appears to be much less likely than with the other two with, however, a lower value of $QDIF$.

5 Conclusion

In this paper, three algorithms for fault-tolerance in mutiprocessor and multicomputer systems have been evaluated and compared. All three algorithms rely on the retriability of a computation task. Two of these algorithms (TMR and DMR) are iterative in nature and the outcome at each retry is independent of the previous trials. The third one (RAFT) is a recursive algorithm, at every retry the result is compared to all the previous ones for that task.

The schemes were evaluated according to three types of measures: (1)the system reliability as expressed by the quality of the decision and the quality of the decision improvement factor, (2) the overhead incurred in both number of retries and total number of processors used and (3) the relative improvement per unit of overhead.

From the performance evaluation of these three schemes we can conclude that, in the general case, TMR provides a good relability. In particular when the reliability of the single processor (p) can be arbitralily low, the TMR algorithm provides a better performance than the other two schemes. When the reliability of a single processor can be expected to be larger than 0.7 then the RAFT and DMR algorithms provide a better reliability than TMR with a lower cost in overhead. Furthermore, it was observed that for these values of p the RAFT and DMR algorithms have the exact same performance levels.

References

[AA86] P. Agrawal and R. Agrawal. Software implementation of a recursive fault-tolerance algorithm on a network of computers. In *Proceedings of the* 13^{th} *Annual Symposium on Computer Architecture*, pages 65–72, 1986.

[Agr85] P. Agrawal. RAFT: A recursive algorithm for fault-tolerance. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 814–821, 1985.

[BF76] M.A. Breuer and A.D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, Rockville MD., 1976.

[KR80] J.G. Kuhl and S.M. Reddy. Distributed fault-tolerance for large multiprocessor systems. In *Proceedings of the* 7^{th} *Annual Symposium on Computer Architecture*, pages 23–30, July 1980.

[KR86] J.G. Kuhl and S.M. Reddy. Fault-tolerance considerations in large multiple-processor systems. *IEEE Computer*, 19(3):56–67, March 1986.

[PF82] J.H. Patel and L.Y. Fung. Concurrent error detection in ALU's by recomputing with shifted operands. *IEEE Transactions on Computers*, C-31(7):589–595, July 1982.

[PMC67] F.P. Preparata, G. Metze, and R.T. Chien. On the connection assignement problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16:848–854, December 1967.

[SS82] D.P. Sieworek and R.S. Swartz. *The Theory and Practice of Reliable System Design.* Digital Press, Bedford, Mass., 1982.

[vN56] J. von Neuman. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, Princeton, NJ, 1956.

	$I(p)$	$QDIF(p)$	$Rit(p)$	$Rir(p)$
TMR	1.00001	33366	11122	33366
RAFT	1.002	99800	49850	99600
DMR	1.002	99800	49800	99600

Table 1: Comparative performance for $p = 0.999$